Does the plate look correct?

SRIVINAYAK CHAITANYA ESHWA, seshwa@gatech.edu
SHERAZ HASSAN, shassan74@gatech.edu
RENCHU WANG, rentruewang@gatech.edu
SACCHIT MITTAL, skmittal@gatech.edu

In the competitive restaurant market, a perfectly plated meal that can be enjoyed with all human senses helps differentiate restaurants by playing a pivotal role in customer satisfaction and service efficiency. Food standardization regarding visual appearance and portion sizes promotes customer loyalty and positive word-of-mouth [1]. However, ensuring consistency in plating and presentation across diverse dishes in a restaurant can take time and effort, with a significant monetary impact. Our computer vision system, accompanied by a mobile application for automated food standardization, to determine if a plate of food in a restaurant adheres to specified standards in terms of presentation and composition, allows restaurants to deliver visually appealing and consistent dishes. From an economic perspective, it can lower operational costs by minimizing the need for manual inspection and re-plating, aiding resource allocation and even menu optimization.

1 INTRODUCTION

With increasing competition in the restaurant market, visual aesthetics and consistency (across multiple locations) are crucial to improving the customer experience and promoting customer loyalty. Setting up manual systems to ensure the same would cause restaurants to incur significant costs. Hence, we worked on a system to automate the process so that the staff could immediately recognize an issue with a plate of food concerning aesthetics and consistency in terms of portions and ingredients.

Our solution consists of a comprehensive system with an iPad application that allows the restaurant staff to take pictures, send them to our servers for evaluation, and display the results to the staff. The results provide the staff with immediate feedback regarding the plate based on visual appearance. To simulate how a restaurant's Quality Assurance (QA) station would capture these images, we used an external camera connected to the iPad.

Real-time detection of the food layout requires a Computer Vision (CV) model that can process and segment images in real-time on the domain of food classification. We also needed to detect ingredients and their positioning to determine where the ingredients are and their relative position. Then, we have to classify which target menu item the dish is. The results must then be communicated to the restaurant staff through the application.

To summarize, we achieved the following milestones

- Trained CV Model for Ingredient Detection
- Menu Item Identification through Ingredients Classification
- Calculation of Similarity Score Using Segmentation Mask
- Creation of an iPad application for seamless interaction

The report is organized with Section 2 providing a background, Section 3 describing the methodology, Section 4 addressing challenges, Section 5 presenting results, Section 6 offering conclusions, and Section 7 outlining future work.

2 BACKGROUND

Multiple datasets [2] and [3] with labeled images are available for food-related tasks. However, these datasets (what we note as "dishes datasets") only provide the names and categories of the entire dish, but not the segmentation of an image and what ingredients/components a dish has. We ended up using UECFOODPIX [4], a segmentation dataset based on FOODPIX [5], which is itself a dishes dataset. This dataset includes pixel-level segmentation labels for 10k with 101 classes of food items.

Since our segmentation model is expected to be fast and run in real-time, we have explored several options. We explored from R-CNN [6], Fast R-CNN [7], Faster R-CNN [8], and YOLO [9], which to this date is still state of the art [10]. Compared to the R-CNN family of models, using YOLO on dataset segmentation has the benefit that most of the work in segmentation is done in machine learning models and thus can be deployed and run on GPU accelerators, significantly speeding up the inference process. In this work, we used YOLOv8 as our backbone for fine-tuning. YOLOv8 is pre-trained on the COCO dataset [11] and is faster and more accurate than its predecessor because of its updated architecture, which is suitable for real-time prediction.

Moving on to the iPad application, we explored card sorting [12] technologies to determine the information architecture. However, due to the lack of access to restaurant staff, we created a prototype iPad application first and used techniques like interviews and cognitive walkthroughs to evaluate it. The latest developments in Apple's AVFoundation API helped us to connect the external camera seamlessly to the iPad application.

3 METHODOLOGY

3.1 Data Pre-processing

We used the UECFOODPIX [4] dataset for our model. It has 10000 images of 101 different food classes. We refer to these food classes as ingredients for our dishes. Since UECFOODPIX is a segmentation dataset with a pixel-by-pixel mask, it is incompatible with the YOLO model as the model requires the input to be a text file with all the polygons of each item preceded by the label of each item. To combat this issue, we converted all the masks of the images into polygons. We then divided the images into a ratio of 9 to 1 for training and validation purposes.

3.2 Model Training

We opted for YOLOv8 [3] as our backbone with Keras-CV [13] for their ease of configuration as the framework for our backend. Among the various options within the YOLOv8 model to get the segmentation mask and classification of the ingredients, we used the nano model to minimize latency, and it has the least number of

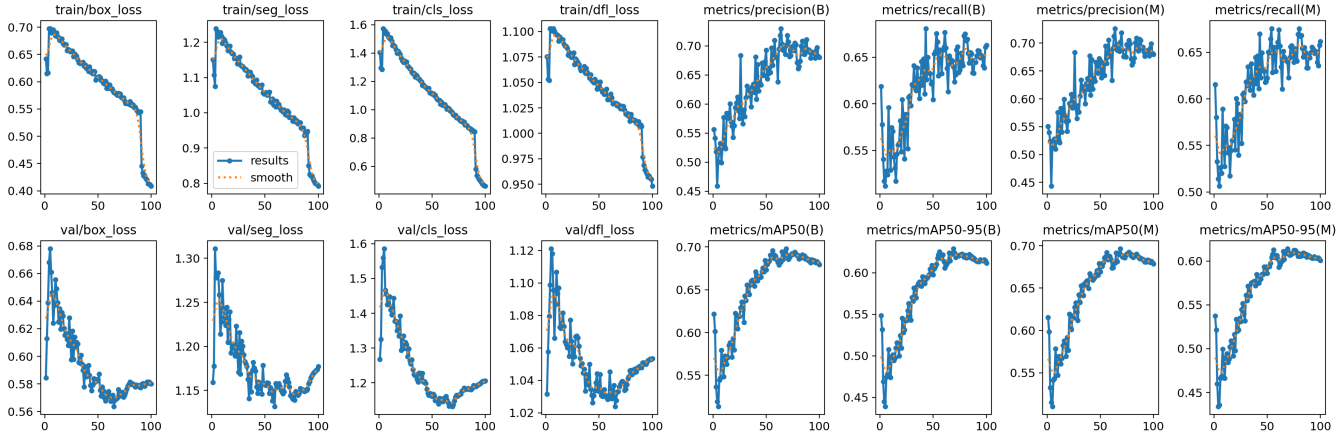


Fig. 1. YOLOv8-nano Training Results on UECFOODPIX

parameters. The loss function employed during training is Mean Squared Error (MSE), represented as:

$$\text{MSE Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

where n is the number of samples, y_i is the ground truth, and $\hat{y}_i$ is the predicted value.

We trained the model on V100 GPU for 100 epochs employing a learning rate of 0.001 and ADAM optimizer. To prevent overfitting, we utilized a checkpoint to get the weights of the model with the minimum segmentation validation loss instead of the last trained model. The training and validation loss of the model across epochs is depicted in Figure 1.

This trained model depicts proficiency in predicting the segmentation mask and classification with bounding boxes of the ingredients in the UECFOODPIX dataset. We used this ability of YOLOv8 to both classify and segment at the same time in our favor to achieve the required results in the post-processing steps.

3.3 Menu Item Prediction

We utilize the classification and bounding box capabilities of YOLOv8 to classify each ingredient within the image individually. This enables us to identify the labels and count of each ingredient in the image. Once we have identified all the ingredients, we refer to the menu dictionary containing the menu items and their corresponding ingredients. Through cross-comparison, we calculate a score indicating how closely the detected ingredients match those in the menu items. The menu item with the highest closeness score is then predicted as the menu item in the image.

$$\text{Confidence Percentage} = \frac{\text{Number of Matching Ingredients}}{\text{Total Ingredients in Menu Item}} * 100 \tag{2}$$

3.4 Similarity Score

We originally planned for a pretraining + finetuning approach, much like many modern deep learning systems. However, we ended up using a similarity-based approach. When an image comes in, and we predict the menu item in the image, we cross-match the segmentation masks of the ingredients with the "standard menu item" image we have in the storage and decide the similarity between each ingredient within the image. The reason for the decision is two-fold:

- We have been in contact with the restaurant late into the semester, where we have started most of our work.
- The restaurant could not provide too many high-quality images and could only provide 1-2 images per menu item.

We first use the YOLOv8 model we finetuned to decide the location/layout of the ingredients on the dish, and then adopt algorithms like SIFT [14] and ORB [15] that are rotational invariant to figure out the best match between the dish and the menu items. However, during experiments, we realized that it was not robust enough. Sometimes, even flipping the image would yield a low similarity score. Hence, we added the open-source version of OpenAI's CLIP model [16] to compute the similarity score as an additional feature.

3.5 iPad Application

We conducted a need-finding exercise using interviews with an executive of the restaurant we were working with and our project mentor. We identified the requirements to build a prototype. We built an iPad application prototype using Apple's Xcode IDE using the Swift programming language. This prototype could connect to an external camera to take images, upload them to the server for evaluation, and display the results once the evaluation was completed. It also had a data collection mode (not connected to our backend), which allows restaurant staff to collect and label images of plates to train our model further. We developed the user interface to get feedback to address any shortcomings.

For the backend, we used Firebase Storage and Firebase Realtime Database along with listeners. The front end is connected to these

storage and real-time databases. Once the front end captures an image, the storage is updated with the image. Through listeners, we can detect these changes in the database. Once these changes are detected, the uploaded image is downloaded on the local machine and fed to the machine learning model. After the model processes the image, it returns a JSON with the output image and the similarity score. This image is posted to the storage, and the database is updated with the new JSON. Again, through listeners, we can detect these changes in the database. Once the change is detected, the frontend pulls the updated JSON, which has the image and similarity score. The JSON is parsed to retrieve the output image and to retrieve the similarity score. Finally, the output image, along with the similarity score, is presented on the User Interface.

4 CHALLENGES

In our class, we aim to create computing services that can run everywhere at users' convenience. We want to make invisible systems that could make visible impacts. In our project, we designed and created a versatile food layout detection system that users interact with with just an iPad and a camera mounted on top of the food tray. When food is ready, and the kitchen has to notify the servers to serve the food, they could, at a button's convenience, check if the food's layout is correct within seconds. In addition, the system is versatile enough that new dishes can be added to it anytime so that restaurants are not hesitant to innovate on the dishes they offer!

However, we faced multiple challenges when assembling our results, starting with the challenges within the model. With our strict requirement on data (food-related, have segmentation), we are unable to find a large amount of data for training purposes. We overcame this with a pre-training + fine-tuning approach where we used the pre-trained YOLOv8 models, which are trained on the COCO dataset [11] and then fine-tuned on our [4] dataset.

Another issue that we faced within the UECFOODPIX dataset was that there was a data imbalance as seen in Figure 2.

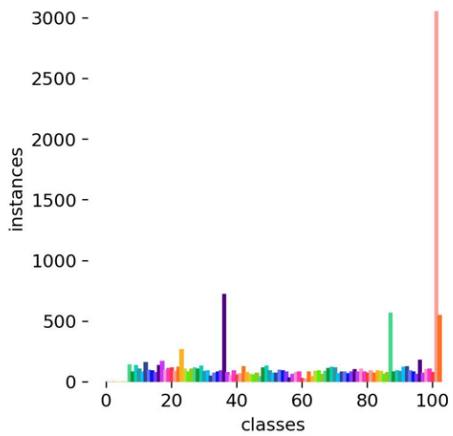


Fig. 2. Distribution of each Class

This data imbalance made it difficult for the model to predict certain classes (1 - 8) with fewer instances, which was reflected in our results during prediction. The solution to this issue was not as simple, however because multiple classes were present in a single image, and we could not just pick and choose certain classes within an image to train the model.

In the next step, when we went to YOLOv8's official repository for the most up-to-date model from the original authors, we realized that the authors did not fully open-source the training part of the model; that is, the open-sourced version is limited to only fine-tuning on a single label, which is insufficient for our purposes. In addition to a completely new set of labels, we also wanted to fine-tune the YOLOv8 models on the multiple labels that we need to detect the layout of the dishes successfully. Thus, we looked to another framework, Keras CV, that provides both the model weights and fine-tuning utilities such that we could handle the training efficiently.

During the app development step, the initial need-finding exercise was delayed due to the availability of the restaurant's executive. Hence, the user interface needed to be modified a little later in the project timeline. We were not able to get access to the restaurant staff, and hence we could not conduct card sorting to inform the information architecture.

5 RESULTS

Fig. 3. Predicted Segmentation Mask and Class Labels

We evaluate our model classification based on precision and recall. Precision is the ratio of correctly predicted positive observations to the total predicted positives and recall is the ratio of correctly predicted positive observations to the total actual positives. Their equations are given by:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4}$$

After training and testing the model, we obtained the following results when evaluating it on 1000 images:

precision(B)	recall(B)	precision(M)	recall(M)
0.68016	0.663	0.67987	0.66151

Here, (M) refers to the "macro" average, which calculates the average performance metric over each class, and (B) refers to the "best" performance metric achieved by the model during the training process. The results in the confusion matrix, shown in Figure 4, indicate overall satisfactory performance.

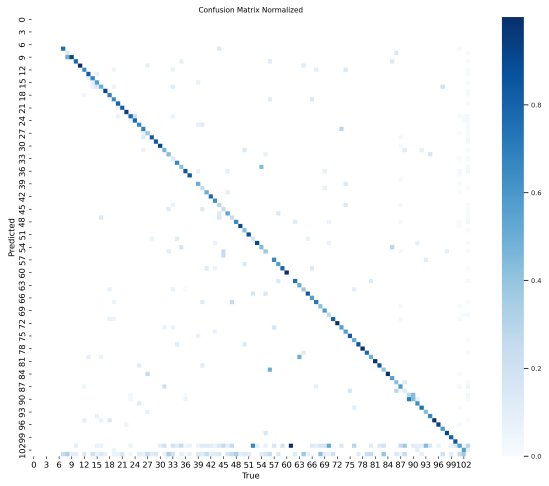
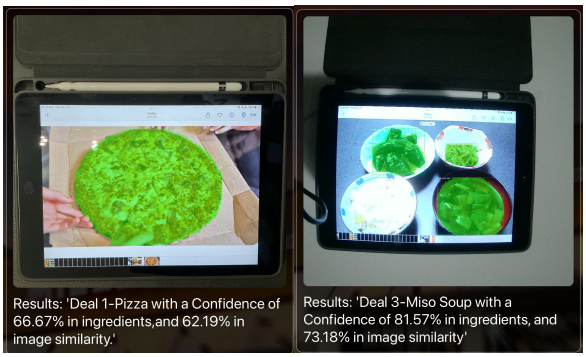


Fig. 4. Confusion Matrix of Results

However, there are noticeable challenges in predicting the first few classes, as observed in the confusion matrix, which is caused by class imbalance, as discussed earlier. Some of the predicted masks and classification labels by our model can be visualized in Figure 3.

The iPad application allowed users to click images of the good items using an external camera and view the evaluated results as shown in Figure 5. The restaurant staff can use this data to make an informed decision on whether or not the plate meets the restaurant's standards.



((a)) ((b))

Fig. 5. Output Displayed on iPad Application

6 CONCLUSION AND REFLECTION

To conclude, we built a fully functional iPadOS app that captures an image of a plated dish and displays the output image and a similarity score on the User Interface. In our process of building out the User Interface, due to a shortage of time and lack of access to the restaurant executive, we could not conduct card sorting. So, we conducted a round of cognitive walkthroughs and an interview with our project mentor to evaluate our interface. We overcame our issue of lack of training data by utilizing a pre-training and fine-tuning approach along with training the model for ingredients rather than training it for menu items, which will also decrease the need for retraining with new menu items included as the new item might be using the same ingredients. For the most part, this worked well for us. Due to reasons beyond our control, we could not work with AWS and had to opt for local hosting of the model and connecting the app and model through Firebase.

In the future, we plan to move away from using listeners to detect changes in the Firebase real-time database, host the application on AWS, and ensure communication between the front end and the model through REST-API. This will help reduce latency and allow for future third-party integration or easier expansion. Capturing the image is done manually, which requires intervention by the restaurant staff. An auto-capture feature can be implemented that automatically detects when a plated dish is kept in front of the camera and takes a picture, which will help the restaurant save time, reduce labor, and streamline its operations. On the Model end, making it easily trainable from within the app for a new ingredient in the menu item is a part of the future goals. Additionally, the model's prediction can be improved by using a more advanced YOLOv8 rather than nano but then pruning the model to decrease the number of parameters and ultimately decrease the latency.

REFERENCES

[1] A. N. Putra, S. P. Anantadjaya, and I. M. Nawangwulan, "Customer satisfaction as a result of combination of food display quality," *Manajemen dan Bisnis*, vol. 19, no. 2, 2020.
[2] S. P. Mohanty, G. Singhal, E. A. Scuccimarra, D. Kebaili, H. Héritier, V. Boulanger, and M. Salathé, "The food recognition benchmark: Using deep learning to recognize food in images," *Frontiers in Nutrition*, vol. 9, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnut.2022.875143
[3] A. Antonov, "Food-11 image dataset," Dec 2019. [Online]. Available: https://www.kaggle.com/datasets/trolukovich/food11-image-dataset
[4] K. Okamoto and K. Yanai, "UEC-FoodPIX Complete: A large-scale food image segmentation dataset," in *Proc. of ICPR Workshop on Multimedia Assisted Dietary Management(MADiMa)*, 2021.
[5] T. Ege and K. Yanai, "A new large-scale food image segmentation dataset and its application to food calorie estimation based on grains of rice," in *Proc. of ACMMM Workshop on Multimedia Assisted Dietary Management(MADiMa)*, 2019.
[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.
[7] R. Girshick, "Fast r-cnn," 2015.
[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.
[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
[10] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of yolo algorithm developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022.
[11] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.
[12] J. R. Wood and L. E. Wood, "Card sorting: current practices and beyond," *Journal of Usability Studies*, vol. 4, no. 1, pp. 1–6, 2008.

[13] L. Wood, Z. Tan, I. Stenbit, J. Bischof, S. Zhu, F. Chollet, D. Sreepathihalli, R. Sampath *et al.*, "Kerascv," https://github.com/keras-team/keras-cv, 2022.
[14] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.
[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.
[16] G. Ilharco, M. Wortsman, R. Wightman, C. Gordon, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt, "Openclip," Jul. 2021, if you use this software, please cite it as below. [Online]. Available: https://doi.org/10.5281/zenodo.5143773